Natural Language Processing

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Session 4

Language Models and N-Grams

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin and many others who made their course materials freely available online.

Session Content

N-Grams Language models

- Language Models and Word Prediction
- N-gram language models
- Evaluating Language models
- Smoothing
- Interpolation and Backoff

N-Grams and Language Models

Word Prediction, the chain rule, and the Markov assumption

Language modelling

A language model assigns probabilities to sequences of words.

For a sentence:

- $P(W)=P(w_1, w_2, \ldots, w_n)$

It can also be used to predict the next word:

- $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

A language model tells us how likely a sequence of words is.

Why does word prediction matter?

Many NLP tasks depend on predicting plausible word sequences

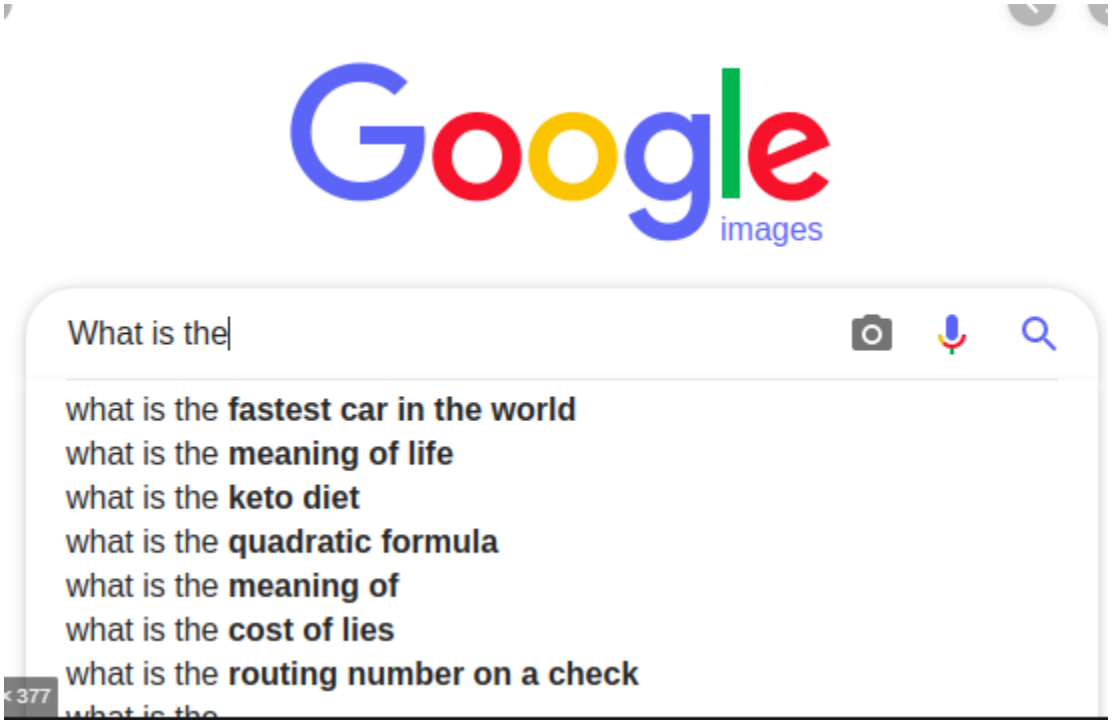
- Spell and grammar correction
- Speech recognition
- Machine translation

It is also the basic idea behind modern LLMs

- Predict the next word/token from the preceding context
- Generate text one token at a time

N-gram models are one of the simplest language models.

Example: Next-word Prediction


Google
images
What is the
what is the fastest car in the world
what is the meaning of life
what is the keto diet
what is the quadratic formula
what is the meaning of
what is the cost of lies
what is the routing number on a check

Application - Machine Translation

Machine Translation

- Machine translation system is used to translate one language to another language
- Language model helps prefers the more natural sentence.
- P(***high*** *winds tonight*) > P(***large*** *winds tonight*)

Application - Spell correction

- The language model assigns a higher probability to the correctly spelled sequences.
- P(*about fifteen minutes from*) > P(*about fifteen minuets from*)

Application - Speech Recognition

- **Speech recognition** is the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format.
- P(*I saw a van*) >> P(*eyes awe of an*)

The Chain Rule

- Recall the definition of conditional probabilities:

  $\mathbf{P}(B|A) = \mathbf{P}(A,B)/\mathbf{P}(A)$

- Hence,

  $\mathbf{P}(A,B) = \mathbf{P}(A).\mathbf{P}(B|A)$

- For multiple variables, the Chain Rule:

  $\mathbf{P}(x_1, x_2, x_3, \ldots, x_n) = \mathbf{P}(x_1).\mathbf{P}(x_2|x_1).\mathbf{P}(x_3|x_1,x_2)\ldots\mathbf{P}(x_n|x_{n-1},x_{n-2}\ldots\ldots,x_1)$

The Chain Rule applied to a Sentence

- $P(w_1 w_2 \ldots \ldots w_n) = \prod_{k=1}^{n} P(w_k | w_1^{k-1})$

For ex:

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

Where:

- P(its)= #(its) / #(*all words*)
- P(water | its) = #(its water) / #(its)
- P(is | its water) =#(its water is) / #(its water)
- P(so | its water is) =#(its water is so) / #(its water is)
- P(the | its water is so transparent) = #(its water is so transparent) / #(its water is so)

The Markov Assumption

Andrei Markov

The chain rule requires us to consider the **entire history** of preceding words.

- That quickly becomes impractical

Simplifying assumption: Predict the next word using only a fixed number of preceding words.

For example:

- $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-k}, \ldots, w_{i-1})$
- $P(\textit{the} \mid \textit{its water is so transparent that}) \approx P(\textit{the} \mid \textit{transparent that})$

The Markov Assumption – Formal Definition

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Only the most recent k words are used to predict w.

N-gram Language models

- An **N-gram** is a sequence of N consecutive tokens (usually words).
- An **N-gram language model** predicts each word using the previous N−1 words.
- The **Unigram** model (N=1)

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

- Example:
- P(*I want to eat healthy food*) ≈ P(*I*).P(*want*).P(*to*).P(*eat*). P(*healthy*).P(*food*)

N-gram Language models

- The **Bigram** Model (N=2)

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

- Predict the current word using one preceding word.

- Example:
- P(*I want to eat healthy food*) ≈ P(I|<start>).P(want|I).P(to|want).
  P(eat|to).P(healthy|eat).
  P(food|healthy).P(<end>|food)

Larger N-gram models

- We can use a **larger N-gram**
  - trigram, 4-gram, 5-gram

- But in general, this is an insufficient model of language
  - because language has long-distance dependencies

  "The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing."

Side Note: Maximum Likelihood Estimates

- The Maximum Likelihood Estimate (MLE), of some parameter of a model M from a training set T:
  - **maximizes the likelihood of the training set T given the model M**
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
  - MLE estimate is 400/1,000,000 = .0004
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

Estimating bigram probabilities

Once we have a bigram model, we need to answer:

- How do we estimate $P(w_i \mid w_{i-1})$ from a corpus?

The **Maximum Likelihood Estimate** **(MLE)**:

- Choose the model parameters that make the observed training data **most likely**.
- For N-gram models, **MLE** reduces to counting occurrences:

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Example: Estimating Bigram Probabilities

Corpus:

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

A larger corpus:
The Berkeley Restaurant Project

Real language data provides the counts needed to estimate N-gram probabilities.

Corpus:

- …
- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day
- …

Step 1: Raw Bigram counts

- Rows = preceding word
- Columns = next word
- Each cell = number of times the bigram occurred

- Out of 9222 sentences (*|V| = 1446*):

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Step 2: Raw Unigram Counts

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Step 3: Raw bigram probabilities

P(i|i) = 5 / 2533 = 0.002

P(want|i) = 827 / 2533 = 0.33

P(to|i) = 0 / 2533 = 0

P(eat|i) = 9 / 2533 = 0.0036

P(i|want) = 2 / 927 = 0.0022

P(to|want) = 608 / 927= 0.66

P(eat|to) = 686 / 2417 = 0.28

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Step 4: Bigram estimates of sentence probabilities

The probability of a sentence is the product of its bigram probabilities.

P(***<s> i want chinese food </s>***) =

P(I | <s>)
× P(want | i)
× P(chinese | want)
× P(food | chinese)
× P(</s> | food)

What does the model learn?

Word Associations:

- P(English | want) = .0011
- P(chinese | want) = .0065
- P(to | want) = .66
- P(eat | to) = .28

Sentence structure:

- P (i | <s>) = .25

And some *zeros*:

- P(food | to) = 0
- P(want | spend) = 0

Practical Issues: Computing Sentence Probabilities

- Sentence Probabilities are products of multiple world probabilities:

$$P(W) = \prod_i P(w_i|w_{i-1})$$

- For long sentences, this could become vanishingly small

- So, we typically operate in log space:

$$\log P(W) = \sum_i \log P(w_i|w_{i-1})$$

  - Avoids numerical underflow
  - (also adding is faster than multiplying)

Why study N-gram models, if we have LLMs?

- N-grams can't handle long-distance dependencies, and don't generalize well to new sequences
  - Large language models handle this better - they use neural embedding spaces and can model meaning
- But n-grams are the ‘fruit fly of NLP’ - a simple model that introduces ideas every language model needs:
  - Training sets, dev sets, and test sets
  - The perplexity metric
  - Sampling to generate text, and generalization
  - Interpolation and backoff
- N-grams are also efficient, fast, and useful when a full LLM would be too heavyweight

innovate achieve lead

BITS Pilani

Pilani Campus

Generalization and Zeros

Why n-grams overfit, and what zero counts do to a model

The perils of overfitting

- N-grams work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't – for instance, the test set could contain new word combinations
  - We need to train robust models that generalize beyond training data
  - One problem is unseen N-grams -> **zero probabilities**

Unseen N-grams: Zero Probabilities

The test set can contain perfectly reasonable word sequences that never appeared in the training set.

- **Training set:**

  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

- **Test set**

  … denied the offer
  … denied the loan

P(“offer” | denied the) = 0

Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!

- If $P(w_i \mid w_{i-1}) = 0$,
  - then any sentence containing this bigram gets $P(W) = 0$

innovate achieve lead

BITS Pilani

Pilani Campus

Smoothing

Laplace (add-1) smoothing, interpolation, and backoff

Laplace Smoothing (*Add-1* smoothing)

- Pretend we saw each word *one more time* than we actually did.
- Just add one to all the counts!

- MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate: $$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$

LaPlace Smoothing
(Berkeley Restaurant Corpus)

innovate achieve lead

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

V = Total number of unique words in corpus = 1446

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigram probabilities

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

V = Total number of unique words in corpus = 1446

```
P(want|i) = 828/(2533+1446) = 0.21
P(to|i)   =   1/3979        = 0.00025
P(eat|i)  =  10/3979        = 0.0025

P(to|want)= 609/(927+1446)  = 0.26
P(eat|to) = 687/(2417+1446) = 0.18
```

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

- A smoothed probability can be expressed as an "adjusted count" divided by the original, unsmoothed context count.

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} = \frac{C^*(w_{n-1}w_n)}{C(w_{n-1})}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Example: c*(i,want) = (828*2533)/ (2533+1446) = 527

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Add-1 smoothing - a blunt instrument

- Add-1 smoothing also substantially reduces the probabilities of events that were observed frequently.
- Add-1 is generally not used for N-gram language models
  - We need better methods to redistribute probability mass
- But Add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.

Using Less Context - Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - Use the highest-order model available; fall back to lower-order models when necessary
  - Trigram -> Bigram -> Unigram
- **Interpolation:**
  - Combine information from multiple orders
  - Trigram + Bigram + Unigram
- In practice, interpolation often gives better performance.

Linear Interpolation

- Combine unigram, bigram and trigram probabilities:
- Simple interpolation

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

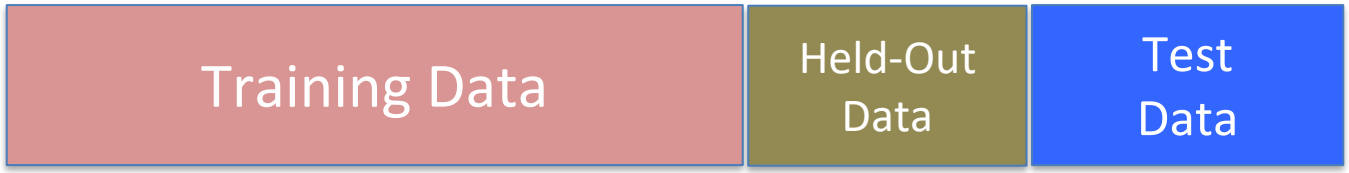
$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1}) P(w_n)$$

Choosing interpolation weights

Use a **held-out** corpus



- **Train** N-gram probabilities on the training set.
- **Keep those probabilities fixed.**
- Use a **held-out(validation)** set to choose λ's.
- Choose the λ's that maximize held-out likelihood.

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Another Generalization Problem: Unknown Words

innovate achieve lead

Closed vocabulary

- Vocabulary V is fixed and known.

Open vocabulary

- Test data may contain words not seen during training.
- These are **out-of-vocabulary (OOV)** words.

For OOV words, create an unknown word token <UNK>

- During training:
  - Create a fixed lexicon L of size V
  - At text normalization phase, any training word not in L changed to <UNK>
  - train its probabilities like a normal word
- During decode:
  - Use UNK probabilities for any word not in training

innovate achieve lead

BITS Pilani

Pilani Campus

The Web and Stupid Backoff

Handling web-scale n-gram data efficiently

Google N-Gram Release: Web-Scale Data

At this scale, storing and processing all possible N-grams becomes a systems problem.

AUG 3

All Our N-gram are Belong to You

...

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects,

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Making Web-Scale N-grams Manageable

[Each N-gram is stored with its observed count]

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html

https://pypi.org/project/google-ngram-downloader/

Huge web-scale n-grams

- **Pruning**
  - Keep only N-grams that occur sufficiently often.
  - For example:
    - C(N-gram)>threshold
- This reduces storage, memory and lookup cost

Stupid Backoff for Web-scale N-grams

- **Stupid backoff** (Brants *et al*. 2007)
  - Use the longest available history. If it isn't available, back off to a shorter history.
  - No explicit probability normalization or discounting - use relative frequencies with a backoff weight.

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4 S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

  - Instead of doing sophisticated probability smoothing, use a simple backoff mechanism that is computationally cheap enough for enormous N-gram models.

innovate achieve lead

BITS Pilani

Pilani Campus

Evaluating Language Models

Train/dev/test sets, intrinsic evaluation, and perplexity

Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
  - Than “ungrammatical” or “rarely observed” sentences
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
  - A **test set** is an unseen dataset that is different from our training set, totally unused.
  - An **evaluation metric** tells us how well our model does on the test set.

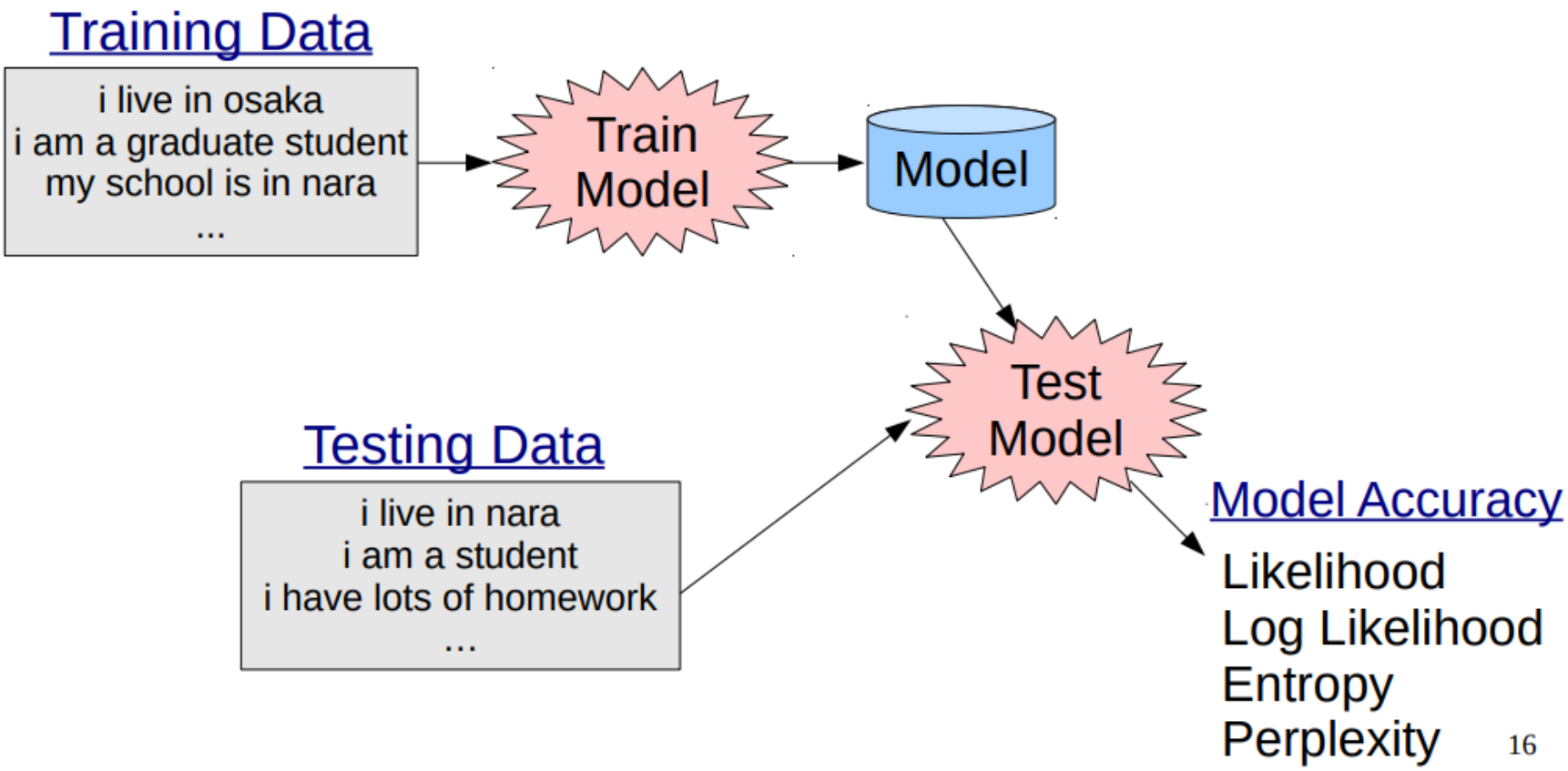
Why do we need a Dev Set

- If we tune our model on the test set repeatedly, we implicitly fit it to that test set

- This means we need a third dataset: a development test set, or devset
  - We evaluate and tune our model on the devset throughout development
  - We touch the test set only at the very end, to report final performance

- Never let test sentences leak into the training set
  - Otherwise the model assigns them artificially high probability, making it look better than it really is
  - "training on the test set" — bad science

Experimental setup



Extrinsic evaluation of N-gram models

- Measures usefulness in **the real application**
  - Spell correction -> correction accuracy
  - Speech recognition -> recognition accuracy
  - Machine translation -> translation quality
- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Intrinsic v/s Extrinsic Evaluation

Extrinsic evaluation

- Realistic
- Directly measures task performance
- Expensive / time-consuming

Intrinsic evaluation

- Fast
- Evaluate the language model directly
- Common metric: **perplexity**
  - If the test data looks **very different from the training/application domain**, perplexity may be a poor approximation of real-world performance.
  - Nevertheless, it is useful for understanding and comparing language models.

Intuition of Perplexity

- **The Shannon Game:**
  - How well can we predict the next word?

I always order pizza with cheese and _____

The 33rd President of the US was _____

I saw a _____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

  - Unigrams are terrible at this game. (Why?)

- A good model assigns high probability to the word that actually occurs.
  - A model that consistently assigns high probability to the actual next word is less "perplexed."

Perplexity

For a sequence of N words, **Perplexity** is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1, \ldots, w_N)^{-1/N}$$

Using the chain rule:

$$PP(W) = \left[\prod_{i=1}^{N} P(w_i|w_1, \ldots, w_{i-1})\right]^{-1/N}$$

For Bigrams:

$$PP(W) = \left[\prod_{i=1}^{N} P(w_i \mid w_{i-1})\right]^{-1/N}$$

Lower perplexity = better next-word prediction.

Perplexity as a branching factor

- The branching factor of a language is the number of possible next words that can follow any word.
- Let's suppose a sentence consisting of random digits 0 to 9 Ex: 0,2,1,3……N
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Perplexity: Comparing N-gram models

- Training set 38 million words, Test set 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

- Lower perplexity indicates better predictive performance on the test corpus.

Common Corpora for NLP

innovate achieve lead

Text corpora

- **Brown Corpus** — general English text
- **Wall Street Journal (WSJ)** — newswire text
- **AP Newswire** — news articles
- **Hansards** — Canadian parliamentary proceedings
- **TRAINS** — task-oriented dialogue
- **Radio News** — broadcast news

Text & speech corpora

- **DARPA/NIST corpora**
- Call Home
- ATIS
- Switchboard
- Broadcast News
- TDT
- Communicator

References

https://books.google.com/ngrams

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

http://www.speech.sri.com/projects/srilm/

https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html

https://www.kaggle.com/code/alvations/n-gram-language-model-with-nltk/notebook

https://levelup.gitconnected.com/bi-tri-and-n-grams-with-python-a9717264e6f2